System-Wide Security for Offline Payment Terminals

Nikolay Ivanov and Qiben Yan(✉)

Michigan State University, East Lansing, MI 48824, USA
{ivanovn1,qyan}@msu.edu

Abstract. Most self-service payment terminals require network connectivity for processing electronic payments. The necessity to maintain network connectivity increases costs, introduces cybersecurity risks, and significantly limits the number of places where the terminals can be installed. Leading payment service providers have proposed offline payment solutions that rely on algorithmically generated payment tokens. Existing payment token solutions, however, require complex mechanisms for authentication, transaction management, and most importantly, security risk management. In this paper, we present VolgaPay, a blockchain-based system that allows merchants to deploy secure offline payment terminal infrastructure that does not require collection and storage of any sensitive data. We design a novel payment protocol which mitigates security threats for all the participants of VolgaPay, such that the maximum loss from gaining full access to any component by an adversary incurs only a limited scope of harm. We achieve significant enhancements in security, operation efficiency, and cost reduction via a combination of polynomial multi-hash chain micropayment channels and blockchain grafting for off-chain channel state transition. We implement the VolgaPay payment system, and with thorough evaluation and security analysis, we demonstrate that VolgaPay is capable of delivering a fast, secure, and cost-efficient solution for offline payment terminals.

Keywords: Blockchain · Off-chain interaction · Smart contract · Offline payment

1 Introduction

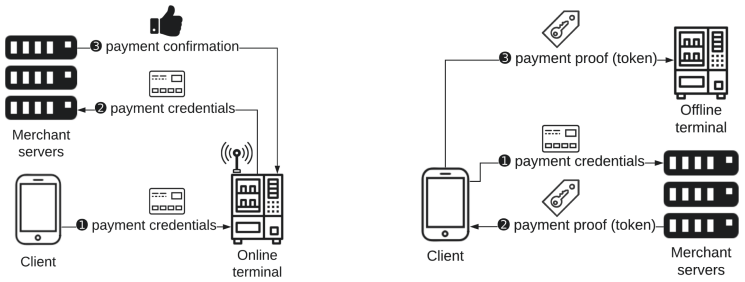
The popularity of electronic payments has grown significantly over the past decade due to increasing number of online users. Consumers got used to online shopping and payments, however, a number of security incidents have undermined their trust in electronic commerce [19,29,32].

Most electronic payment terminals have to stay online in order to process payments. The online requirement, however, is associated with cybersecurity threats, increased costs, dependency upon third-party infrastructure, and limited locations where the terminals can be installed. Offline payment designs

J. Garcia-Alfaro et al. (Eds.): SecureComm 2021, LNICST 399, pp. 99–119, 2021.
https://doi.org/10.1007/978-3-030-90022-9_6

deliver solutions to these problems [7,14,18,26,28,30], which generally require a payment token to be transmitted between the payer and payee through Near-Field Communication (NFC), Bluetooth, electromagnetic field [20], QR code [22] or even audio signal [2].

Figure 1 illustrates the payment transaction workflows in payment systems with online and offline terminals. An online terminal gathers client's payment credentials, and uses them for authentication and payment validation at the merchants' server. By comparison, a payment transaction with an offline terminal allows the client to request the payment token from merchant servers directly and use it for payment verification on the offline terminal.



(a) Online terminal (b) Offline terminal

Fig. 1. Payment transaction workflows with online and offline terminals.

However, one major challenge of designing offline payment infrastructure is to securely manage user balance, including over-spending and double-spending prevention. In traditional payment channels, an offline terminal cannot be timely updated on changes of a user balance, and thus it is subject to token lifting and spending attacks [4]. One way to address this challenge is to use cryptographic payment tokens (i.e., cryptographic proofs of payment) that a user delivers to the payment terminal. Yet, the token-based payment systems face four major concerns: security, privacy, trust, and cost-inefficiency.

First, in the classic payment token designs, the user balances and transactions are stored in a centralized database, which is a single point of failure. The resilience of a payment token infrastructure can be enhanced at the expense of storing multiple copies of encryption and signature keys on different nodes. Second, the sensitive client information, such as credit card number, often has to be stored in the merchant infrastructure, which is notorious for being a target of massive cyberattacks [19]. Third, the classic payment token designs further assume the trustworthiness of the merchants, while the necessity to protect clients from potentially malicious or fake merchants is largely ignored. Fourth, it requires a significant effort and investment to ensure the security and resilience of a classic solution. As a result, the cost and complexity of the payment token infrastructure may outweigh the benefits of using offline terminals.

In this paper, we design a novel payment system, VolgaPay, for token-based offline terminals. The design of our system relies on the following two insights:

1. We use blockchain and cryptocurrency, which provide an effective solution for eliminating otherwise unnecessary reliance of client upon merchant's trustworthiness. We supplement the blockchain technology with a multi-hash chain-based micropayment channel and polynomial price representation in order to achieve a balance of security and efficiency for payment transactions between merchant and client.
2. To achieve high efficiency, enhanced security, and operation cost reduction, we propose the *blockchain grafting* technique to conjoin a fast, zero-fee auxiliary smart contract in a private blockchain with a decentralized smart contract on a public blockchain.

VolgaPay operates offline points of sale (OPOS) and uses multi-hash chain micropayment channels and *blockchain grafting* to enhance the security and efficiency of offline payment transactions. Each merchant-client association is represented by a novel combination of smart contracts: one smart contract deployed on the public blockchain (public smart contract), and another one deployed on the private blockchain (private smart contract). The public smart contract provides protection of client's deposit and merchant's payoff without any pre-existing trust, while the private smart contract provides secure interface and accounting to the off-chain micropayment infrastructure of the merchant with minimum delays and zero fees.

We implement a VolgaPay prototype system using Raspberry Pi-based terminals, Android clients, Ethereum-based public and private blockchains, and a cohort of independent token signer servers scattered across the globe. **A video demonstration of VolgaPay operations is available at** https://youtu.be/rjIhDD2yi5I.

In summary, our contributions are as follows:

- We propose a blockchain-based transaction network, VolgaPay, to address the elevated security risks, inefficiency, and high cost of existing payment systems with offline terminals;
- We implement VolgaPay, which we thoroughly evaluate to determine its low cost, high speed, and impressive scalability: the token request delay normally does not exceed 12 s, the one-time blockchain fee of establishing a smart contract is less than $1, the communication overhead is light enough to serve 3G-connected clients, and the number of simultaneous token requests can reach 10,000.
- We analyze the security of VolgaPay, and prove that: if any component of the system is compromised, the harm will be localized, and the system at large remains secure.

2 Related Work

Off-Chain Payment Channels. The Bitcoin Lightning Network protocol [24] uses Bitcoin smart contract functionality for securing off-chain micropayment

channels. To perform off-chain operation, Bitcoin Lightning uses two-way multi-signature channels based on digital signature schemes, supported by a network of multiple dedicated servers. Currently, there are a few test implementations of the Lightning network, including Lightning Labs [17], Casa Node [5], ACINQ [27], Blockstream [12], and MIT lit [16]. Other cryptocurrency micropayment projects include Perun [8], Revive [15], Sprites [21], Bolt [11], SpeedyMurmurs [25], Stellar [1], and a duplex micropayment channel [6]. Blockumulus [13] uses a cloud-based smart contract, called *FastMoney*, for off-chain payments. The functionality and security of all the solutions above largely depend on the setup and performance characteristics of centralized or crowd-sourced servers that serve the off-chain network. To avoid security risks and synchronization complexity associated with centralized micropayment management, VolgaPay uses *blockchain grafting* for transaction management. Moreover, the aforementioned solutions are not designed to support payments with offline terminals.

Offline Electronic Payments. There are a few solutions proposed for offline electronic payments. The most popular schemes involve delivering a cryptographically verifiable token from client to terminal using a local offline channel, e.g., electromagnetic or radio transmission [14]. Other solutions propose the use of trusted hardware [9,18], which relies on the assumption of hardware integrity. While trusted hardware is useful in a deeply regulated and controlled environment, such as Apple Pay, it could not support anonymous payments in a low-trust environment, e.g., a self-service station.

3 System Design

We elaborate on the design of VolgaPay by describing its fundamental definitions and principles, as well as three classes of elements constituting its topology: *participants*, *communication channels*, and *protocols*. Figure 2 shows the overall system architecture, in which each transaction between client and merchant is supported by a public smart contract supplemented by a linked (grafted) private smart contract. The public smart contract protects client's deposit and refund, while guaranteeing payoff for the merchant. The private smart contract provides secure interfacing endpoint for the client and secure accounting and ledger-keeping for the merchants' micropayment channels. The client pays the merchant with a set of hashes, and receives a payment token (i.e., verifiable payment receipt), which is a tuple of signatures from several independent signers that respond to the token request blockchain event. Table 1 summarizes all notations used in the design of VolgaPay.

Polynomial Multi-Hash Chain Micropayment. Traditional blockchain-based micropayment channels rely on digital signature schemes, which require the protection of private keys and prevention of double-spending complicated by transaction reversals, whereas hash chain-based micropayment channels eliminate the use and storage of private keys and avoid transaction reversals. In this

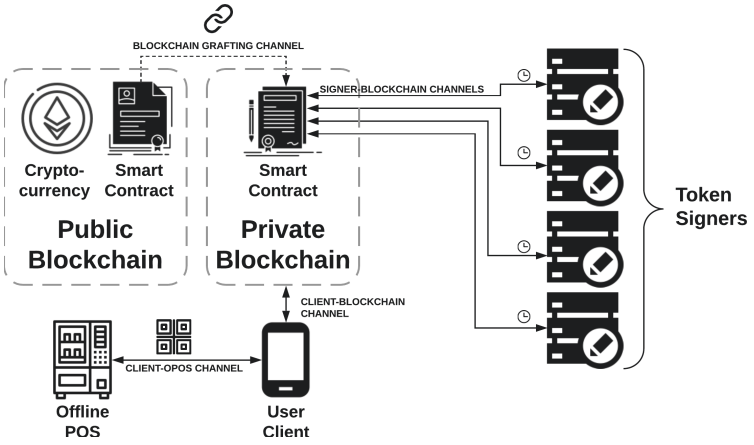


Fig. 2. VolgaPay system architecture. The ⏲ label symbolizes an event-listening channel. VolgaPay includes seven types of participants and four types of channels. OPOS first delivers the information needed for payment and token request to the client through the client-OPOS channel. The client then provides payment and requests payment token from the private blockchain through the client-blockchain channel. Signers subscribe to the token request event via the signer-blockchain channels. Private smart contract extends the public smart contract through a virtual blockchain-grafting channel. Finally, the client delivers a payment token to OPOS using the client-OPOS channel.

Table 1. Notation table for VolgaPay.

Symbol	Meaning
η	Transaction ID randomly generated by OPOS
δ	Item's listed price or single payment amount
D	Human-readable item description
N	Total number of token signers
M	Number of signatures in a token
$\Gamma_i(m)$	Digital signature of message m signed by token signer i
$K(m)$	Digital signature of message m signed by an OPOS
$E(p)$	Private blockchain event with parameter p
τ	Price base—the sum of previous payments.
ϕ	τ-adjusted price (*cumulative payment*): $\phi = \tau + \delta$
Θ	Hash chain polynomial base (max. number of hashes in each hash chain)
Υ	Ordered sequence of payment hash chains $\Upsilon = (\Upsilon_1, \Upsilon_2, ..., \Upsilon_\xi)$
ξ	Number of hash chains in Υ
Υ_i	i-th hash chain in Υ
R_i	The number of hashes released by the client to the merchant from Υ_i
g	Price granularity measured in number of atomic units

paper, we use hash chain-based micropayment channel for VolgaPay's off-chain transactions. The client generates the hash chains and keeps their seeds in secret. However, the use of hash chain-based micropayment channels involves significant computational overhead. Specifically, long hash chains, which have been successfully used in non-blockchain micropayments [3,23], incur large delays, intolerable fees, and execution timeouts when used in non-view blockchain smart contract calls[1]. Our pre-evaluation of hash chain generation performance on Ethereum Mainnet has demonstrated significant performance degradation and fee increase for non-view verification of Keccak256 and RIPEMD-160 hash chains whose length exceeds 100 hashes[2].

Traditional currencies, such as U.S. dollar (USD), have a coarse granularity (i.e., *cent*) compared to the ultra-fine granularity of most cryptocurrencies, e.g., in Ethereum, 1 *wei* $= 10^{-18}$ *Ether*. Moreover, if we assume the granularity of 1 *cent*, the traditional hash chain-based micropayment channel, in which one hash represents one atomic payment unit, would require to produce 10,000 hashes for the verification of a 1,000-*dollar* payment, which is practically infeasible on blockchain due to its significant computational overhead.

To address this problem, we propose a *polynomial multi-hash chain micropayment scheme* which utilizes several hash chains to process arbitrary micropayments. Instead of using a very long single hash chain, in which each hash represents a minimal payment unit, such as Ethereum's *wei*, we use a ξ-tuple of short hash chains, each responsible for a single digit with radix Θ. For example, to account for USD prices between 1¢ and \$10,000 (or 10^6¢), we need 7 decimal digits, i.e., $\xi = 7$, $\Theta = 10$, with 7 hash chains and 10 hashes in each chain. Each digit from 0 to 9 can be represented by one of the 10 hashes, totalling 70 hashes for any price in the given range. In comparison, 10^6 hashes are required in the classic single hash chain representation. Therefore, any single payment amount δ can be represented as the following polynomial:

$$\delta = \sum_{i=1}^{\xi} R_i \times \Theta^{i-1}, \tag{1}$$

where R_i is the number of hashes released from hash chain Υ_i. For instance, suppose the client pays \$25.31 (or 2,531¢), the micropayment will be executed by releasing 1 hash from Υ_1 ($R_1 = 1$), 3 hashes from Υ_2 ($R_2 = 3$), 5 hashes from Υ_3 ($R_3 = 5$), and 2 hashes from Υ_4 ($R_4 = 2$), while $R_5 = R_6 = R_7 = 0$.

If *wei* is used as the payment unit, we need 19 hash chains to represent prices up to 9 *Ether*, as 1 *Ether* $= 10^{18}$ *wei*. The number of hash chains can be reduced by increasing the granularity g and capping the maximum cumulative off-chain balance. To process multiple payments, we submit the τ-adjusted price

[1] In smart contracts, a "view" function is a read-only function of smart contract that does not modify the state of blockchain. The "view" functions can be executed by the blockchain API layer rather than miners. Thus, the execution is generally fast.

[2] On Mainnet, the fee for 100-hash verification exceeds 80¢ with recommended gas price. The attempt to verify 270 hashes fails with "out of gas" error.

(or cumulative payment) ϕ instead of the single payment δ, and we use the price base τ to track the previous payments corresponding to already released hashes. Thus, we adapt Eq. (1) to include g and τ. For each payment, the client determines a sequence of *hash chain depths* $(R_1, \ldots, R_\xi)$, to express ϕ as:

$$\phi = \tau + \delta = g \times \sum_{i=1}^{\xi} R_i \times \Theta^{i-1}. \tag{2}$$

Increasing granularity g reduces the number of hash chains in the set, and thus reduces the public blockchain fees. In our prototype, we use $g = 10^{13}$, which can represent a range of prices between 0.00001 and 99.99999 *Ether* using just 7 hash chains with 10 hashes for each chain.

Example: Although VolgaPay uses *Ether*, we focus on USD in our examples for ease of illustration. Suppose we want to represent USD prices up to \$999,999, with granularity $g = 10^2$, i.e., we round the cost to the nearest dollar. Thus, the granularity-adjusted number of hash chains ξ is 6 rather than 8.

Suppose the client makes three consecutive payments: \$1,720, \$56, and \$56. Assume there are no previous payments in the channel (i.e., $\tau = 0$), then for the first payment, $(R_1, R_2, ..., R_7)$ is $(0, 2, 7, 1, 0, 0, 0)$. For the second payment ($\delta = 56$, $\tau = 1,720$, $\phi = 1,776$), the 7-tuple hash chain depth sequence will become: $(6, 7, 7, 1, 0, 0, 0)$. Comparing the two sequences of sets, we can see that in the second payment, the client reveals 6 hashes from Υ_1 and 5 more hashes from Υ_2. For the third payment ($\delta = 56, \tau = 1,776$, $\phi = 1,832$), the 7-tuple hash chain depth sequence is $(2, 3, 8, 1, 0, 0, 0)$. Compared to the previous payment, the client releases 1 more hash from Υ_3. It is worth noting that after three payments, the number of hashes known by the merchant for each hash chain is $(6, 7, 8, 1, 0, 0, 0)$, worth \$1,876. A malicious merchant can try to get payoff of \$1,876 (rather than \$1,832), which will be rejected by payoff verification as illustrated in Sect. 3.3.

3.1 VolgaPay Participants

VolgaPay system includes seven classes of participants described below.

Public Blockchain. Public blockchain is used for storage, access, execution, and integrity assurance of the public smart contract. VolgaPay design requires the public blockchain to be universally available, trusted by all parties, capable of executing Turing-complete smart contracts with integrated cryptocurrency.

Public Smart Contract Public smart contract is used in VolgaPay to establish an unambiguous and non-repudiable agreement between the merchant and the client. Unlike the deposits in classic electronic payment systems, smart contract allows the client to retain full control over deposited funds irrespective of the merchant's trustworthiness. Each merchant-client pair requires a deployment of at least one separate public smart contract.

Table 2 describes a minimal set of routines that the public smart contract should have. The *set heads* routine, which saves all hash chain heads at once, can only be called by the merchant. The *payoff* routine exchanges the hashes released by the client into the cryptocurrency funds deposited by the client earlier. The *refund* routine allows the client to request unclaimed funds after a certain time period. The *deposit* routine allows the client to fund the smart contract.

Table 2. Minimal set of routines in VolgaPay public smart contract.

Routine	Description	Access
Set heads	Store hash chain heads provided by client	merchant
Payoff	Transfer funds to merchant based on cryptographic proof	merchant
Refund	Request refund after a payoff or timeout	client
Deposit	Deposit funds on the smart contract	client

Private Blockchain. Unlike its public counterpart, the private blockchain provides security only for the merchant infrastructure, and is not intended to be trusted by the client. Managed by the merchant and executed under a proof-of-authority (PoA) consensus, the private blockchain achieves high levels of security and redundancy for the merchant payment infrastructure at very little cost. Unlike proof-of-work (PoW) consensus, in which the voting power is restricted by computational limits, the PoA consensus limits the voting power by hard-coding the public keys of the sealers in the genesis block. PoA substitutes mining with sealing, and therefore achieves a better performance and full control over consensus. As an option, several merchants can share one private blockchain based on mutual trust. Alternatively, a merchant can use a third-party PoA blockchain as a service delivered by a trusted provider[3].

Private Smart Contract. For each public smart contract, the merchant deploys a mirroring smart contract on its private blockchain. The public smart contract contains an address of the mirroring smart contract in the private blockchain. We call this technique *blockchain grafting*[4].

Blockchain grafting supplements the public smart contract with the private one, the benefits of which include: first, private smart contract can perform secure execution without paying any blockchain fees; second, private smart contract executes and confirms transactions much faster than its public counterpart; third, private smart contract provides locally-trusted execution and storage within multiple nodes of the merchant infrastructure, such that if one or even several nodes

[3] Our VolgaPay prototype uses Kovan Testnet as a service.

[4] Grafting is an agricultural technique, in which a part of one plant is conjoined with another plant in order to combine the benefits of both. Similarly, we join public and private blockchains to utilize the properties and strengths of the two.

are compromised, the integrity of the ledger remains intact; fourth, private smart contract can achieve secure and flexible redundancy management at low cost: nodes can be added, removed, replaced or updated without service disruption.

Table 3 describes the minimal set of routines provided by the private smart contract. The *set heads* routine mirrors its public blockchain counterpart: it sets the hash chain heads to exactly the same values as they are in the public smart contract. The *token request* routine verifies the legitimacy of the OPOS and client, checks the hashes and payment amount provided by the client, processes the payment, updates the list of revealed hashes, and finally triggers a blockchain event that activates the token signers. The *one-time deposit* routine funds a smart contract only once (using a control variable); the additional deposits are prohibited to avoid tampering with the private smart contract balance. The *set* Γ_i routines are used by the signers to store their signatures in the blockchain.

Table 3. Minimal set of routines for VolgaPay private smart contract.

Routine	Description	Access
Set heads	Store hash chain heads provided by client	merchant
Token request	Process micropayment transaction and generate signatures	client
One-time deposit	Deposit pseudo-cryptocurrency equal to off-chain balance	merchant
Set Γ_i	Save signature $\Gamma_i(\eta, \delta)$ in the smart contract	signer i

Token Signers. To achieve security redundancy, we define the *payment token* as a set of M public-key signatures $\{\Gamma_1(\eta, \delta), ..., \Gamma_M(\eta, \delta)\}$ produced by M out of N signers. The set of signers' public keys is pre-determined by the merchant and stored in all offline terminals. Each token signer is a process running on a physically-separated hardware. All signers are listening to a smart-contract event triggered by the *token request function* of the private smart contract. This configuration allows the signers to communicate with the blockchain using a pull-only protocol, without listening on any ports.

Offline Point of Sale. The offline point of sale (OPOS) stays offline during the normal operation. Each OPOS stores its own private key and the full list of signers' public keys. Each OPOS has its own list of merchandise and pricing policy. If the list of signers' keys is modified in the system, e.g., a new signer is added to the system for extra redundancy, this information must be manually updated on each OPOS. No other change in the merchant infrastructure requires updating OPOS. If OPOS is compromised, i.e., its private key is stolen, the corresponding public key should be removed from or blacklisted in the private smart contract.

Client. VolgaPay client is a mobile device, which can establish a client-OPOS channel with an OPOS in proximity. The client also has access to the merchant's private blockchain through a client-blockchain channel. For purchases, the clients do not access the public blockchain.

3.2 Communication Channels

VolgaPay has four classes of communication channels: blockchain grafting channel, client-OPOS channel, client-blockchain channel, and signer-blockchain channels, as shown in Fig. 2.

Blockchain Grafting Channel. The blockchain grafting channel virtually links the address spaces of two independent blockchains. Specifically, it connects the public and private blockchains by storing the address of the private smart contract in the public smart contract.

Client-OPOS Channel. VolgaPay client establishes a local bi-directional simplex channel with an OPOS in proximity. We assume the channel has limited capacity, possible low bandwidth, and is not persistent. For our prototype, as described in Sect. 4, we use two-way QR-code scanning as client-OPOS channel. However, alternative channels can be established, including electromagnetic, Bluetooth, etc.

Client-Blockchain Channel. The client-blockchain channel allows the OPOS to get necessary updates on the state of the micropayment channel. While OPOS remains offline, the client plays the role of a proxy to the merchant infrastructure, through which the OPOS receives a receipt of successful payment in the form of a verifiable token. For the most common use cases of VolgaPay, e.g., vending machine purchase, it is necessary for the client to have access to the merchant infrastructure while interacting with OPOS.

Set of Signer-Blockchain Channels. Each signer is an independent network node subscribed to the token request events of all the private smart contracts through a signer-blockchain channel. This arrangement allows the signers not to listen to any ports and remain anonymous, and therefore significantly limit the exposure to potential cybersecurity threats. In order to serve multiple clients simultaneously, the signer-blockchain channels must maintain sufficient available bandwidth, which is discussed in Sect. 5.4.

3.3 VolgaPay Protocols

VolgaPay protocols define communication procedures between the participants of the system. VolgaPay includes four major protocols: 1) *contract initiation and deposit*, which establishes relationship between client and merchant; 2) *transaction protocol*, which describes the procedure of making transactions; 3) *payoff and refund*, which defines the conversion of the micropayment channel state into a fiat cryptocurrency; and 4) *payoff verification*, which verifies the correctness of the payoff amount. None of the four protocols require the client to share any sensitive or personal information; thus, data privacy and identify anonymization are inherently guaranteed by our blockchain-based design.

Contract Initiation and Deposit. This protocol includes the following sequence of steps:

1. The client requests a contract from the merchant, e.g., through the merchant's website or using a mobile app, and sends to the merchant the heads of hash chains in the set Υ;
2. The merchant prepares a pair of smart contracts for the client—public and private, and sends the client the address of the public contract;
3. The client verifies the code of the public smart contract, extracts the address of the private smart contract, and deposits funds on the public smart contract.

Transaction Protocol. The purchase protocol includes the following steps:

1. A client approaches an OPOS and initiates a purchase, e.g., by selecting an item with description D and price δ from the list of merchandise, as exemplified in Fig. 3;
2. The OPOS generates a random unique transaction ID η, and delivers the tuple $(\eta, \delta, D, K(\eta, \delta))$ to the client;
3. The client converts the item price δ into the τ-adjusted price ϕ, using the variable τ stored in the private smart contract;
4. The client calls the *token request function* of the private smart contract, providing $(R_1, R_2, ..., R_\xi), \eta, \delta$, and $K((\eta, \delta))$ as arguments;
5. The merchant's private smart contract first verifies price, payment, balance, and the authenticity of the signature of the OPOS, and then stores the tuple of payment hashes in the smart contract. It updates the smart contract balance, and adds current payment amount to the sum of previous payments τ, which is stored in the private smart contract. Finally, it triggers the event $E(\eta, \delta)$;
6. Each token signer responds to the event $E(\eta, \delta)$ by storing the signature $\Gamma_i(\eta, \delta)$ in the private smart contract;
7. The client waits until the private smart contract accumulates a minimum required number of signatures, and then it serializes these signatures into a tuple (which is the payment token), and delivers this tuple to the OPOS;
8. The OPOS verifies whether the signatures are distinct, valid, and produced by legitimate token signers, and then delivers the purchased item to the client.

Payoff and Refund. The merchant first verifies the payment hashes in the private smart contract released by the client, and then calls a payoff function of the public smart contract with the inputs of payment hashes, payment amount, and optional client's payoff verification signature (as described below). The unused funds deposited in the smart contract can be delivered to the client as a refund. The implementation specifics of payment and refund are determined by the merchants' policies.

Payoff Verification. VolgaPay's polynomial multi-hash chain price representation creates a potential caveat for the merchant to learn more hashes from the hash chains than the desirable number of hashes determined by the value of ϕ, as described in Sect. 3. VolgaPay enforces the correctness of the payoff amount by adding a payoff verification signature, $sig(\phi)$, generated and signed by the client to authorize an exact transaction payment amount. The signature will be stored as an additional variable in the private smart contract and updated after each

successful token request. The signature and corresponding ϕ will be verified by the public smart contract during the payoff process: if the merchant's requested payment deviates from the correct payment on the blockchain, the payoff will be subsequently rejected. We leave the implementation of this feature as future work.

4 System Implementation

We have implemented the VolgaPay payment system. VolgaPay prototype includes two independent OPOS emulators and two Android clients. The total size of implementation includes 6,100+ lines of code. The details of different parts of the prototype are discussed below.

Fig. 3. OPOS interface transition.

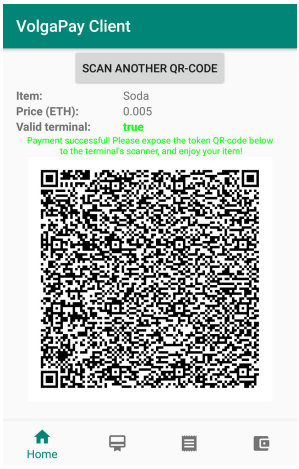


Fig. 4. VolgaPay Android client.

Offline Point of Sale (OPOS). We simulate two OPOS using two sets of Raspberry Pi 3B+ as computing modules, mini-touchscreen for user interaction, and Raspberry Pi camera for reading QR-codes. We use Python 3.6, TkInter, and Web3.py to implement the OPOS software. Both the terminals are offline.

Figure 3 shows a three-step user interface transition of a purchase. The first step is the selection of the item: when the user pushes the touchscreen button, the terminal generates a unique transaction ID, and produces a QR-code. The QR-code contains the transaction ID η, item price δ in *wei*, item description D, and signature of the tuple (η, δ). During the second step, the client scans the QR-code with the Android app, requests a token, produces QR-code for the token, and pushes the "Scan Token" touch button on the screen of OPOS, which

activates the QR-code reader. Finally, when the QR-code is scanned, the OPOS verifies the signatures, and delivers the item to the client.

Android Client. We implement a VolgaPay Android client using Android SDK and Web3j. Figure 4 shows the screen of the Android client right after obtaining a payment token from the private blockchain. Curious readers please refer to our demo video https://youtu.be/rjIhDD2yi5I for more information.

Mainnet Smart Contract. The public smart contract, which we test on both Ropsten and Mainnet networks, uses preset hash chain heads for executing payoff to the merchant and refund to the client. For our prototype, we use the following parameters: base currency–Ether, $\Theta = 10$, $g = 10^{13}$, $\xi = 7$, which means that we use 7 10-hash-deep chains, with minimal price unit equal to 10 *Szabo*, and maximum supported contract balance of 99.99999 *Ether*.

Kovan Smart Contract. We use Kovan Testnet for VolgaPay prototype to simulate the private blockchain. **The code of the private smart contract, written in Solidity, is available through this link:** https://github.com/seitlab/volgapay/blob/main/smartcontract.sol. The `requestToken()` function performs the validation of client identity, the authenticity of the OPOS, the validity of the payment, and available balance. If all the parameters are verified, the private smart contract decreases the balance of the smart contract, updates the base price τ, updates the payment hashes, and emits a signer-activation event.

The Signers. In our evaluation, we use a network of 10 signers. To demonstrate that the signers are not sensitive to latency and performance, we deploy them on low-tier basic servers (1 CPU/ 1 GB RAM/25 GB HDD/Ubuntu $18.04.2 \times 64$), located all across the world. Our prototype requires 5 valid signatures to form a payment token, i.e., $M = 5$. The event listeners are written in JavaScript using Web3.js library. When the token request event is issued by the smart contracts, all the ten signers respond by signing the serialized tuple of transaction ID and item price, and write the signatures in the smart contract.

5 Evaluation

Using the prototype, we perform evaluation of VolgaPay based on three criteria—delays, communication overhead, and cost of fees. Our prototype is not optimized for production use. Further improvement is possible through software optimization, choice of hardware, data compression techniques, network configuration, and selection of cryptographic routines. In the evaluation, each OPOS is implemented on a separate Raspberry Pi Model 3B+ [10]; each signer has the following configuration: 1 CPU, 1 GB RAM, 25 GB HDD, Ubuntu 18.04.2. As clients, we use Huawei Mate 9 and Motorola Moto G5 Plus, both with Android Nougat.

5.1 Delays

Here, we measure two types of delays, which determine the feasibility of VolgaPay: token request time and payment request delay on public blockchain.

Transaction Latency of Traditional Payment Methods. We conduct a field experiment to measure a payment transaction latency of different types of traditional self-service terminals using Chase VISA debit card and CapitalOne MasterCard credit card, using both magnetic stripe and RFID technologies. We made 2 purchases at 2 different self-service gas stations, 4 purchases at 3 vending machines, 2 purchases at two parking meters, 2 balance checks, 2 cash withdrawals, and 1 PIN verification at 2 ATMs belonging to 2 different banks. We measure the transaction delay between applying the payment and receiving a payment acknowledgement using a handheld stopwatch. In order to adjust for human response delay, we subtract 1 s from each reading rounded to the closest second. The results, shown in Table 4, exhibit a significant variance, with the transaction times ranging from 5 to 21 s, with standard deviation of 5.3, and mean average of 11.7 s.

Token Request Time. We evaluate the token request time by measuring the time delay in milliseconds between pushing the "Get Token" button in the client Android App and the appearance of the token QR-code on the screen of the client. We measure the delay under three different types of Internet connection of the client: fast WiFi, LTE, and 3G. For each of the three connection types, we perform 20 probes, and record the mean average, standard deviation, and the speed profile of each connection[5] in Table 4. The experiment shows the average token request time, ranges between 9 and 12 s.

Table 4. Token request delays with different client connections. The transaction latency of VolgaPay is comparable to the credit card transaction delays with traditional payments (ranging from 5 to 21 s in our field experiment).

Client connection	Avg. link bandwidth			Delay (ms)	σ
	Down	Up	Ping		
WiFi	39.67	18.02	10.5	9,399	1,837
LTE	13.57	12.99	26	10,026	1,769
3G	2.54	0.90	80.5	11,618	1,716
Traditional self-service with credit cards	—	—	—	11,700	5,300

[5] For speed profile, we measure the speeds using Ookla Speed Test and M-Lab Speed Test, and get their average.

Public Smart Contract Calls. We measure the delays of public smart contract calls on Ropsten Testnet and Etheretum Mainnet networks. Each measurement is repeated 10 times, and the average results recorded in Table 5. The results show that the Ropsten network, despite its similarity with the Mainnet, does not always produce similar execution delays. The measurements also show that all the smart contract calls complete within 30 s on average. The results prove that the VolgaPay transactions introduce reasonable delays.

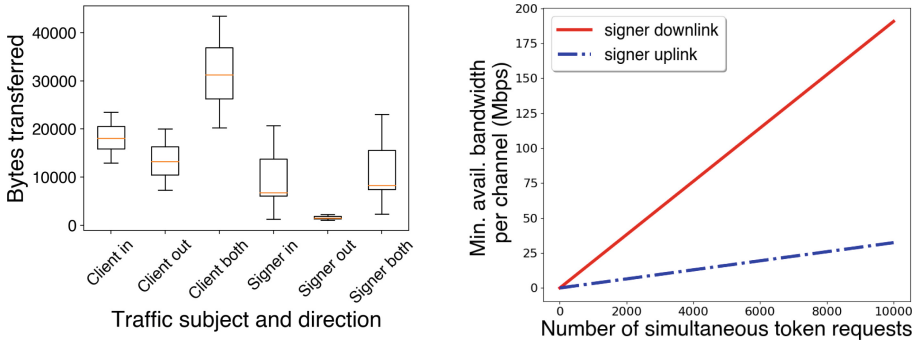
Table 5. Average public smart contract delays on Ropsten and Mainnet networks using Infura API, based on 10 measurements, with gas price 2.2 Gwei.

PHCS operation	Ropsten testnet		Ethereum mainnet	
	Delay (ms)	σ	Delay (ms)	σ
Bytecode check	756	371	10,920	698
Balance check	79	28	700	12
Deposit	28,168	13,874	22,555	20,823
Set 7 hash chain heads	26,467	11,547	26,491	25,222
Retrieve hash chain heads	489	41	554	65
Retrieve stored public key	175	46	236	205
Payoff	23,333	28,338	21,413	16,019
Refund	20,031	15,400	27,898	25,529
Contract deployment	24,481	10,913	20,231	9,623

5.2 Communication Overhead

Token Request. In our prototype, we use Infura API for communication with Kovan network. The summary of the communication overhead through the client-blockchain channel over 30 measurements is summarized in the left half of Fig. 5a. The result shows that the overall communication overhead does not exceed 50 Kbytes, and the incoming traffic volume is slightly larger than the outgoing one. The right half of the graph delivers the summary of communication overhead over signer-blockchain channels, which shows the total overhead per signature rarely exceeds 20 Kbytes, and the incoming traffic volume is also much larger than the outgoing one.

QR-Codes. The ASCII-based QR-code produced by the OPOS in our prototype includes the following fields: transaction ID, price in *wei*, description, and signature. The total length of the sequence encoded in the OPOS QR-code ranges between 175 and 222 bytes. The length of the second QR-code (i.e., payment token) is determined by the parameter M (i.e., the number of required signatures) of the system. For VolgaPay prototype, we use $M = 5$ with `0x`-prefixes and 1-byte delimiters (for error checking), allotting to 664 bytes per token.



(a) Avg. token request communication overhead over 30 measurements.

(b) Min. avail. bandwidth requirement per signer-blockchain channel.

Fig. 5. VolgaPay communication overhead and associated throughput.

5.3 Fees

Table 6 shows the gas fees of VolgaPay prototype's public smart contract implementation over 10 measurements on Ropsten and 10 measurements on Mainnet network. As shown in the table, the operations are cheap, and the most gas-consuming one is contract deployment. Note that smart contract deployment is a one-time cost, which is less than $1.

Table 6. Gas fees, based on 20 measurements.

Smart contract operation	Average gas used	σ	Approximate USD cost[a]
Set addresses	21,040	0	$0.012
Set heads	72,263	65	$0.040
Payoff	55,212	53	$0.030
Refund	30,001	0	$0.017
Contract deployment	922,915	60,545	$0.508

[a]With the gas price 2.2 Gwei.

5.4 Scalability

Number of Simultaneous Token Requests. Let us assume that the available bandwidth of all signer-blockchain links is approximately the same, and the

maximum time allotted for one signature response equals the average time T_b (in seconds) of sealing one block in the private blockchain. Then, the minimum available bandwidth B_{min} for one signer (in bps) is $B_{min} = \frac{N_R \cdot L}{T_b}$, where L is communication overhead per request in bits, and N_R is the number of simultaneous token requests. Using this formula and the communication overhead measurements gathered from the VolgaPay prototype (see Fig. 5a), we can demonstrate (see Fig. 5b) that a channel with 100 Mbps (200 Mbps) bandwidth is able to support more than 5,000 (10,000) simultaneous token requests.

Number of Clients. The number of clients in VolgaPay is not limited by any practical parameter of the system. However, it is important to note that as the number of clients increases, the probability of exceeding the bandwidth capacity of the signer-blockchain channel grows.

Number of OPOS Terminals. The number of OPOS terminals affects two parameters in the private smart contract: the set of OPOS public keys, and the amount of computation required to verify the identity of a terminal. We modify the smart contract by adding 10^3 random OPOS keys, which increases the number of OPOS identity checks from 2 to 1,000 in the token request routine. Then, we re-evaluate the token request delay by running 20 more token requests simulating the worst-case scenario. The results indicate similar delays, which do not exceed 10 s on average. Therefore, for each merchant, VolgaPay can support at least 1,000 OPOS terminals without degrading performance.

6 Security Analysis

Here, we account for the possible reasonable scenarios in which important components (participants or channels) are compromised by an adversary. If several types of attacks exist for achieving the same malicious outcome, we elaborate on the outcome, e.g. "gaining full access".

Malicious Merchant. Malicious merchants try to steal funds from the clients. Note that the clients' funds in public smart contract are guarded by the public blockchain and their respective private keys. The only feasible malicious activity would be to receive payment without providing a valid token, in which case the client will lose payment for a single purchase. Given the cost of infrastructure setup, the merchants are unlikely to pursue such attacks.

Full Access to OPOS. Each OPOS stores its own private key, which is used for confirming its belonging to a certain merchant. If the key is stolen, it can be temporarily used for the deployment of a fake OPOS, but it does not allow the attacker to steal any funds since OPOS does not process payments. Therefore, the incentives to steal the private key of an OPOS is limited only to retaliation and vandalism. In order to protect the private key from theft, it is recommended to store the key only in the address space of RAM of the legitimate VolgaPay process. Additionally, if an attacker gains full physical access to an OPOS, all the merchandise of the terminal can be stolen bypassing the token verification

procedure. This, however, can be prevented by enforcing physical security, which goes beyond the scope of this work.

Full Access to the Client App. The client app can be designed not to store the private key used in public smart contract, or to store it securely encrypted, with mandatory passphrase solicitation before each use. As a result, we exclude the possibility of stealing the private key used for public smart contract even when the adversary gains access to the client software. Thus, the only valuable asset that can be stolen from the client is the set of hash chain heads, and the authentication private key for the private smart contract: the combination thereof can be used to make purchases on behalf of the client, but does not allow to steal funds because fund management is performed via the private key of the public smart contract.

Full Access to One of the Private Chain Sealers. If one of the sealers of the private blockchain is compromised, there is no immediate threat to the system. Moreover, depending on the specifics of the private blockchain's consensus protocol and the size of the clique or quorum, the blockchain remains intact even if several sealers are compromised. In most cases, it requires to compromise 50% of the sealers to attack the private blockchain. However, a compromised sealer likely means one of the private keys used for the consensus is stolen. Although there is no immediate danger, the sealer must eventually be replaced, which will require creating a new genesis block, restarting blockchain, re-deployment of private smart contracts, and changing the grafting links in corresponding public smart contracts, which can be done automatically during a scheduled maintenance.

One way to mitigate the threat associated with compromised private blockchain sealers is to use reputable outsourced PoA blockchain services, or to share blockchain between several mutually trusted merchants. Another solution is to add additional IP authentication to the boot nodes, which grants access to the blockchain's P2P network only to peers from certain IP addresses; this solution, however, requires to run a sufficient number of boot nodes in order to prevent an eclipse attack [31].

Denial of Service Attacks. Although denial-of-service attacks do not yield any immediate fund gain for an attacker, they can be used as a means of retaliation or vandalism. Private blockchain sealers and VolgaPay signers are possible targets to this type of attack. The design of VolgaPay allows to keep the IP addresses of both signers and sealers in secret; and if this information still gets revealed, to seamlessly replace the IP addresses without changing blockchain-level identities (keys). This is possible because sealers in blockchain are identified by their keys, not by their IP addresses.

Local Channel Sniffing and Spoofing. Among many ways of maintaining the client-OPOS channel, the QR-code exchange is the most feasible and popular approach. The visual channel is subject to eavesdropping attacks, and its payload limitations make it infeasible to secure, e.g., through a Diffie-Hellmann key exchange. Thus, we can assume that the information in the channel can be read (sniffing) and substituted (spoofing). Because of the physical presence of

the client at the OPOS for the entire duration of transaction, the sniffing or spoofing of any of the two QR-codes does not yield any gain to the attacker other than creating a minor annoyance to the client.

Man-in-the-Middle-Attacks. The resistance to man-in-the-middle (MITM) attacks in all the VolgaPay channels, except the client-OPOS channel described above, can be achieved by establishing encrypted communications. Since most of the channels, such as secure socket or RPC-calls to blockchain, are secure by default, the possibility of such an attack is very low. However, even if an MITM attack is successfully conducted, the potential harm can only be experienced by the client, whose payment token could be blocked by an attacker. There is little incentive for an attacker to do so, as the token validity is limited by the current transaction session.

7 Conclusion

In this paper, we presented a new blockchain-based tokenized payment system, VolgaPay, to address the numerous practical challenges towards the deployment and use of offline payment terminals, such as security, privacy, trust in merchant, and significant infrastructure cost. VolgaPay incorporates multi-hash chain-based micropayment channels and blockchain grafting to strike a balance in security and efficiency, by leveraging the security and trustworthiness of public blockchain, as well as the speed and cost-efficiency of private blockchain. We implemented the VolgaPay payment system and evaluated all the parameters affecting its practical feasibility. Our evaluation shows that the system is fast, cost-efficient, and scalable. Most importantly, through a comprehensive security analysis, we demonstrated that VolgaPay is resistant to a variety of cyber-attacks: if any component of the system is compromised, the scope of harm will be minimized, and the threat will not propagate to other components.

Acknowledgement. We would like to thank the anonymous reviewers for providing valuable feedback on our work. This work was supported in part by National Science Foundation grants CNS1950171 and CNS1949753.

References

1. Stellar (2019). https://www.stellar.org/. Accessed 23 May 2019
2. Alipay: Qr code payment (2017). https://docs.open.alipay.com/140/104622/. Accessed 25 July 2019
3. Anderson, L., Cushley, S.: System and method of providing tokenization as a service (Nov 28 2017), US Patent 9,830,595
4. Bai, X., et al.: Picking up my tab: Understanding and mitigating synchronized token lifting and spending in mobile payment. In: 26th USENIX Security Symposium (USENIX Security 2017) (2017)
5. Casa: Casa node (2019). https://keys.casa/lightning-bitcoin-node. Accessed: 23 May 2019

6. Decker, C., Wattenhofer, R.: A fast and scalable payment network with bitcoin duplex micropayment channels. In: Pelc, A., Schwarzmann, A.A. (eds.) SSS 2015. LNCS, vol. 9212, pp. 3–18. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21741-3_1
7. Dmitrienko, A., Noack, D., Yung, M.: Secure wallet-assisted offline bitcoin payments with double-spender revocation. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, pp. 520–531 (2017)
8. Dziembowski, S., Eckey, L., Faust, S., Malinowski, D.: Perun: virtual payment hubs over cryptocurrencies. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 327–344 (2019)
9. Dziembowski, S., Faust, S., Hostáková, K.: General state channel networks. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 949–966 (2018)
10. Foundation, R.P.: Raspberry pi 3 model b+ (2019). https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/. Accessed 26 July 2019
11. Green, M., Miers, I.: Bolt: anonymous payment channels for decentralized currencies. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 473–489 (2017)
12. Inc., B.C.: Blockstream lightning network (2019). https://blockstream.com/lightning. Accessed 23 May 2019
13. Ivanov, N., Yan, Q., Wang, Q.: Blockumulus: a scalable framework for smart contracts on the cloud. In: 2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS). IEEE (2021)
14. Jiang, F., Okonkwo, A.P., Aitenbichler, E.: Secure offline payment system (Oct 1 2015), US Patent App. 14/226,785
15. Khalil, R., Gervais, A.: revive: Rebalancing off-blockchain payment networks. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 439–453 (2017)
16. Lab, M.M.: layer 2 — the lightning network (2019). https://dci.mit.edu/lightning-network. Accessed 23 May 2019
17. Lightning Labs, I.: Lightning labs (2019). https://lightning.engineering. Accessed 23 May 2019
18. Lind, J., Naor, O., Eyal, I., Kelbert, F., Pietzuch, P., Sirer, E.G.: Teechain: reducing storage costs on the blockchain with offline payment channels. In: Proceedings of the 11th ACM International Systems and Storage Conference (2018)
19. Manworren, N., Letwat, J., Daily, O.: Why you should care about the target data breach. Bus. Horiz. **59**(3), 257–266 (2016)
20. Mendoza, S.: Samsung pay: tokenized numbers, flaws and issues. In: Proceedings of Black Hat USA, pp. 1–11 (2016)
21. Miller, A., Bentov, I., Kumaresan, R., McCorry, P.: Sprites: Payment channels that go faster than lightning. arXiv preprint arXiv:1702.05812 (2017)
22. Nseir, S., Hirzallah, N., Aqel, M.: A secure mobile payment system using qr code. In: 2013 5th International Conference on Computer Science and Information Technology, pp. 111–114. IEEE (2013)
23. Pedersen, T.P.: Electronic payments of small amounts. In: Lomas, M. (ed.) Security Protocols 1996. LNCS, vol. 1189, pp. 59–68. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-62494-5_5
24. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments (2016)

25. Roos, S., Moreno-Sanchez, P., Kate, A., Goldberg, I.: Settling payments fast and private: Efficient decentralized routing for path-based transactions. arXiv preprint arXiv:1709.05748 (2017)
26. Sabba, Y., Scheinfield, J.: Token check offline, 4 August 2016. US Patent App. 15/011,366
27. SAS, A.: Acinq (2019). https://acinq.co. Accessed 23 May 2019
28. Tebbe, H.: Offline mobile payment process (2015), US Patent App. 13/966,053
29. Trautman, L.J., Ormerod, P.C.: Corporate directors' and officers' cybersecurity standard of care: the yahoo data breach. Am. Univ. Law Rev. **66**, 1231 (2016)
30. Van Damme, G., Wouters, K.M., Karahan, H., Preneel, B.: Offline nfc payments with electronic vouchers. In: Proceedings of the 1st ACM Workshop on Networking, Systems, and Applications for Mobile Handhelds, pp. 25–30 (2009)
31. Wüst, K., Gervais, A.: Ethereum eclipse attacks. Technical report ETH Zurich (2016)
32. Xu, W., Grant, G., Nguyen, H., Dai, X.: Security breach: the case of tjx companies, inc. Commun. Assoc. Inf. Syst. **23**(1), 31 (2008)